Real-Time Tasks Scheduling With *Value* Control to Predict Timing Faults During Overload

Crineu Tres, Leandro Buss Becker
Federal University of Santa Catarina
Automation and Systems Department
Florianopolis, Santa Catarina, Brazil
{crineu, lbecker}@das.ufsc.br

Edgar Nett
Otto-von-Guericke Universität Magdeburg
Institute for Distributed Systems (IVS)
Magdeburg, Germany
nett@ivs.cs.uni-magdeburg.de

Abstract

Modern real-time applications are very dynamic and cannot cope with the use of worst case execution time to avoid overload situations. Therefore scheduling algorithms that are able to prevent timing faults during overload are required. In this context, the value *parameter has become useful to add generality and flexibility to such systems. In this paper we present the scheduling algorithm called DMB (Dynamic Misses Based), which is capable of dynamically changing tasks* value *in order to adjust their importance according to its timing faults rate. The main goal of DMB is to allow the prediction of timing faults during overloads and thereby support a dynamic tuning of tasks fault rate. It is used to enhance the features of the previously defined TAFT (Time-Aware Fault-Tolerant) scheduler. Obtained results show that DMB in conjunction with TAFT reached the most promising results during overloads, allowing to control tasks degradation in a graceful and determined way.*

1 Introduction

Mobile embedded systems are typically used in safety-critical applications with real-time constraints. They usually interact with a dynamically changing environment, so that motion and sensing activities must be processed online. Examples can be found in factory automation, team robotics, and even in future road-traffic systems [5, 1].

The particular scheduling problem in such scenarios relates to the data-dependent and widely varying execution times of the computational tasks. Measurements in a RoboCup scenario showed that the execution times are not only dependent on the quantitative amount of input data to be processed but also on its semantics which is very much situation-dependent [10]. Due to energy and weight constraints, there is an additional need for highly effective scheduling of limited resources as CPU and communication bandwidth. Therefore, scheduling policies based on Worst Case Execution Times (WCETs) are not adequate. Due to the inherent overestimation of the required computational resources it may imply unacceptable and unnecessary reject decisions of the scheduler.

Becker et. al. [1] have presented an efficient scheduling strategy using EDF algorithms, formalizing the adopted task model and the underlying scheduling mechanism. The TAFT (Time Aware Fault Tolerant) scheduling policy allows scheduling tasks with dynamically changing execution times and still achieves a predictable real-time behavior [6, 9, 8]. In overload situations, this approach so far still exhibits some weaknesses - no longer the complete execution of all tasks can be guaranteed. Applying the EDF algorithm does not allow giving explicit preference to those tasks, which are most critical w.r.t. preemptions. In most of the considered applications it is required that at least some tasks have to be completed in due time even in overload situations.

This paper aims to enhance previous works by proposing additional solutions to problems that can be summarized as follows: (i) how to enable the real-time scheduler in overload situations to dynamically give preference to tasks with higher *values*; (ii) how to provide a concept of dynamic task values such being able to adapt to dynamic changes in the application status to be controlled. Therefore it is developed the scheduling algorithm called DMB (Dynamic Misses Based), which in conjunction with TAFT allows to achieve the mentioned goals.

The rest of the paper is divided as follows: section 2 presents the related works and details TAFT; section 3 presents the proposed DMB algorithm, one of the main contributions of this work; moreover, section 4 describes the performed experiments, presents and discuss the obtained results; finally section 5 draws the obtained conclusions and the future directions of this research.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

2 Related Works

2.1 Value-Based Scheduling

The meaning of importance has been widely discussed by the research community over the last two decades. The *value* parameter was introduced by Jensen et. al. [7] in order to reflect the benefit obtained from finishing a collection of services. It also defines the concept of Time/Utility Functions (TUFs) as an elaborated way of defining the *value*. Clark [4] argues that the *value* parameter is a suitable solution to increase applications flexibility.

Buttazzo et. al. [3] present a study aiming to find the best scheduling algorithm in overload conditions. The *value* parameter was used to evaluate results emphasizing that the best algorithm to be applied does depend on the workload. It is suggested that a further improvement could be obtained if the system could change its scheduling strategy depending on the current workload. Burns et. al. [2] argues that using *value*-based scheduling is a reasonable solution to increase real-time systems flexibility, and also made clear the importance of choosing a correct *value* (or *value* function), since a wrong choice can invalidate the benefits of its use.

2.2 Brief Overview of the TAFT Approach

The *TAFT (Time-Aware Fault-Tolerant)* scheduler [6, 9, 8] was defined to cope with dynamic real-time applications where task execution times vary considerably. Every task is designed as a *TaskPair (TP)* with a common deadline (D_{TP}): a TP constitutes a *MainPart (MP)* and an *ExceptPart (EP)*, thus reflecting the fault-tolerance aspect. The minimal functionality of the EP is to ensure that the respective TP leaves **a)** the controlled application in a fail-safe state and **b)** the controlling system in a consistent state. From the point of view of the scheduler, both parts are treated as separate scheduling entities having their individual timing parameters. The TAFT scheduler always guarantees the completion of either the MP or the EP before D_{TP}.

The Main Part provides the intended functionality of the module. MPs should be scheduled according to a strategy that maximizes the processor utilization and that keeps a high level of MP completion. Therefore, more realistic *Expected-Case Execution Time (ECET)* are allocated to them (while EPs use the WCET). It is now possible that some instances of the MP require more than the allocated ECET to be completed; that is, they may be subject to timing faults or misses. This can have two consequences: First, as the resources allocated to the MP are not sufficient, D_{TP}, the deadline of the respective TP, could be missed. Secondly, if the MP does consume more than the allocated resources, it may steal resources from other TPs, which then will possibly also miss their deadlines (domino effect).

TAFT avoids both these problems. It keeps the MP's timing faults from propagating cross TP's boundaries, such that timing faults in one TP cannot cause timing faults in another TP. To keep TPs from missing their deadlines, TAFT aborts the Main Part of a TP and triggers the EP when it detects that a MP is about to miss the deadline. To address the second problem, when a MP has consumed the allocated ECET the scheduler sets it to the lowest level possible.

TAFT implementation uses the *Latest Release Time (LRT)* algorithm (or reverse-EDF) to schedule the EPs, representing the first-level scheduler. As the second-level scheduler it uses the normal EDF.

In Becker et. al. [1], TAFT is applied in a RoboCup scenario, where task execution times exhibit a huge variation. Simulation results using execution times obtained from RoboCup scenarios confirm that using TAFT it is possible to achieve higher utilization levels. Nevertheless, it is observed that so far it does not provide sufficient support regarding which tasks to prioritize in overload situations, just like most existing real-time scheduling algorithms.

2.3 Compared Algorithms

The scheduling algorithms evaluated along this study are mostly the same ones used in the previous works that form the basis of this paper [3, 6, 8]. Besides EDF, used here for performance comparison, the other algorithms are the High Value First (HVF) and the High Density First (HDF).

HVF is an algorithm that only takes into consideration the *value* for assigning priorities, so tasks with the higher *value* have the highest priority.

HDF is a mix of the previous algorithms. Priority is result of V_i/c_i, where V_i means the *value* and c_i the remaining WCET needed to complete the execution of task T_i. Tasks with higher density will have highest priority.

A new specific algorithm is proposed in this work given that the previously mentioned ones do not take timing faults into consideration and thereby look like less suitable to reach the proposed goals.

3 Dynamic Misses Based Algorithm

The Dynamic Misses Based (DMB) algorithm is defined to provide a controlled and predictable task execution in overload situations. Therefore it takes into account both the task *value* and the percentage of timing faults to produce the schedule. This implies a new manner to assign tasks priorities, which is shown in equation 1 bellow:

$$P_i = V_i \times (k_1 + k_2.MD_i) \quad (1)$$

where P_i is the priority, V_i the *value* and MD_i the percentage of timing faults of task T_i; k_1 and k_2 are constants.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

The parameter V_i indicates the *value* of the task for the system; it can be either static or dynamic, depending on the kind of TUF that is assigned for it. In situations where this parameter is static and there are no timing faults, than DMB behaves just like the HVF scheduler.

The term MD_i can be interpreted differently according to the kind of system that DMB is applied for. In traditional systems, where timing faults are not handled, it represents the rate of missed deadlines. On the other hand, in TAFT scheduler it represents the rate of EPs executions.

The constants in equation 1 allows fine adjustments in the algorithms' behavior. k_1 defines the degree of influence that the timing faults will have in the final priority; lower values will be more easily influenced by little changes in the term MD_i, and vice-versa. The constant k_2 determines how sensitive the algorithm reacts to changes in the timing faults rate. The higher its value, the more sensitive it is.

With constants k_1 and k_2 assuming the value of 1, equation 1 can be simplified into equation 2, which behavior is summarized as follows: while tasks have no timing faults, schedule is calculated only according to V_i (like HVF); as timing faults are notices, the task's priority is increased until twice of its original value.

$$P_i = V_i \times (1 + MD_i) \quad (2)$$

4 Evaluation

To facilitate the performance comparison with the results presented by Becker et. al. [1] and to generalize the obtained experimental results it was decided not to use specific tasks from the RoboCup application but a synthetic task set. Just like the mentioned work, here it is also adopted the set of tasks and workloads proposed by the *Hartstone Benchmark* [11], using execution time variations derived from our RoboCup scenario. More precisely, this analysis uses the *PN (non harmonic)* tasks series. The tasks are periodic, preemptable, and do not compete for resources.

Performed simulations only consider overload scenarios, where traditional schedulability analysis (based on WCET) would fail given that the nominal utilization factor in use already starts at 120% and goes above 185%. However, the metric used in our experiments is the effective utilization, which is calculated after the simulation according to the real execution time of the tasks. The assumed range varies from 85% to 150%. All performed experiments were simulated for something equivalent to a 30 seconds period. Workload increase can be achieved either by using incremental execution times or with the addition of new tasks.

4.1 Results With Constant *Values*

The schedulers chosen for evaluation were tested in a first moment as plain schedulers, i.e., by their own and without any relation with TAFT scheduler. Afterwards they are used as the second-level scheduler from TAFT (in charge of scheduling the MPs). Our analysis focus on observing the number of timing faults for each task, meaning deadline misses if TAFT is not used or, if so, EP occurrences.

Although a similar analysis was performed by Buttazzo et. al. [3], except for the DMB algorithm, the plain experiments were repeated for three main reasons: 1) to simulate a task set that is closer to a real application; 2) to focus on the effective instead of the nominal utilization; 3) to analyze the performance of the individual tasks along the simulation in contrast to the overall performance.

Initially it is analyzed the individual behavior from tasks in each scheduler. As known by the community, plain EDF performs in an "all or nothing" basis, where tasks present almost no timing faults before 100% utilization and almost no successful completions after that. Toghether with TAFT there is a consideble performance improvement, once the domino effect is avoided.

On the other hand, the algorithms HVF and HDF seem to be immune to overload, given that there is no considerable performance decrease along execution. Nevertheless, both algorithms begin to discard the lower priority tasks as the overload increases, making space left for the remaining tasks. In conjunction with TAFT the behavior presented by HVF and HDF is better, but the trend remains similar.

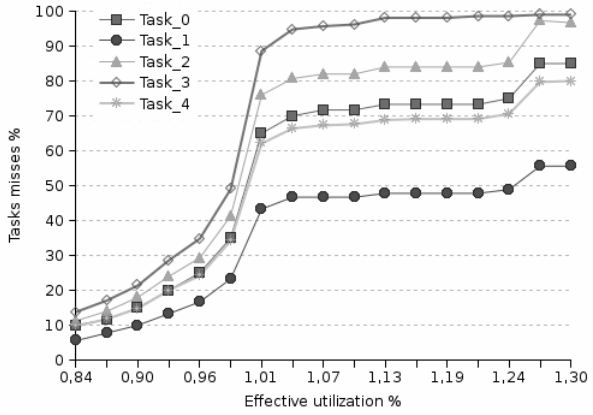

Figure 1. Plain DMB algorithm

The individual analysis of the proposed DMB algorithm is presented in Figure 1. It can be observed that DMB suffers the impact caused by overload (when utilization gets closer to 100%), though slightly better than EDF. However, it is important to notice the expected uniform degradation from all tasks, as proposed in the algorithm specification. Another eminent feature is the inversely proportional relation between tasks degradation and *value*.

In conjunction with TAFT it is observed that DMB reached the most promising results for the established goal

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

of balancing the EPs occurrences among the TPs. This result is shown in Figure 2, which presents tasks having linear increase of EPs execution percentage. For a closer look on the results, the graph y-axis is limited by the 50% mark.

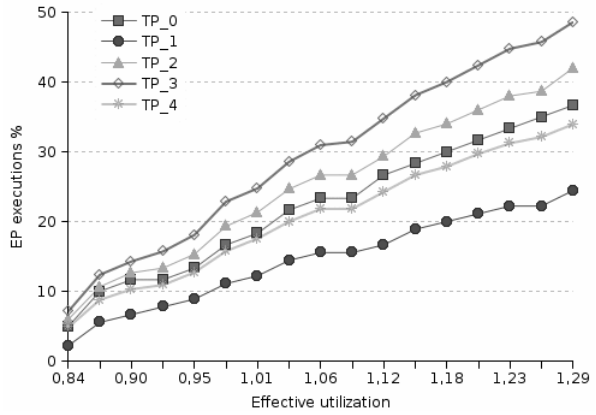

Figure 2. DMB in conjunction with TAFT

Finally, analyzing two distinct TPs scheduled with DMB it is possible to establish that the relation between the percentage of timing faults is inversely proportional to the relation between their *values*. This characteristic is found within any two TPs, which means that the absolute number of EPs executions cannot be controlled (it depends on the effective utilization), but the percentage of EPs executions from one TP relatively to any other TP can be controlled.

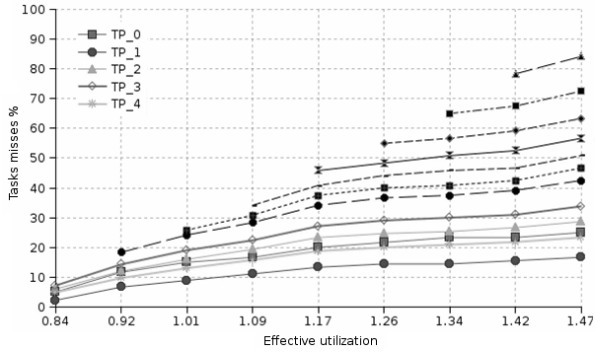

Figure 3. Influence of adding new tasks

To confirm the good results obtained with the DMB scheduler in conjunction with TAFT policy additional experiments were performed. Figure 3 shows the performance of DMB in a system with the same initial tasks as seen before, but now with increasing effective utilization caused by the addition of new tasks (all with the same basic parameters) along the time. Every task, including the initial ones, present constant utilization and at each time interval a new task, with a smaller *value* than the previous one, is added to the system. This makes evident why it always presents the worst results. This experiment is important to confirm the features of the DMB algorithm, as we can see by analyzing the relationship between *value* and timing faults presented by each task. Nevertheless, it is important to highlight that these capabilities are independent of the number of tasks in the system.

4.2 Results With Dynamic *Values*

Here it is considered the use of the DMB algorithm with TAFT and dynamic *values* along execution, looking for an on-line adjustment of priorities. The purpose of this simulation is to assure the system ability to control tasks during execution.

The desired reaction with the incorporation of dynamic values looks like the concept of execution "modes", described by Burns et. al. [2], where tasks functionalities can be performed at distinct levels of quality, thus presenting different requirements.

The simulated scenario is as follows: a system is facing a transient overload, with the task set showing an effective utilization level always above 100%. The simulation is observed during 20 time intervals. Figure 4 shows what would happen if the *value* parameter stays fixed along simulation.

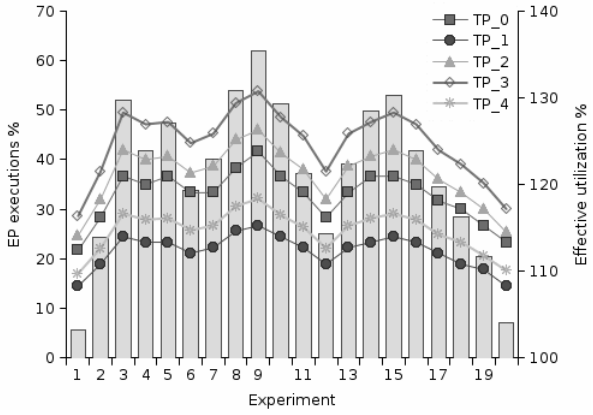

Figure 4. Evaluation with constant *value* parameter

Left y-axis displays the percentage of EP execution by tasks, while right y-axis displays the effective utilization along simulation (represented by the vertical bars). These two statistics overlapped confirm the statement made: the absolute number of EPs executions cannot be previously determined, because it depends on the system utilization. The simulation objective is to change the tasks *value* in order to decrease the percentage of timing faults of specific *TaskPairs*: for example, the two TPs with the worst performances, `TP_2` and `TP_3`. Thus, these TPs must display the best performance results at the end of the simulation.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

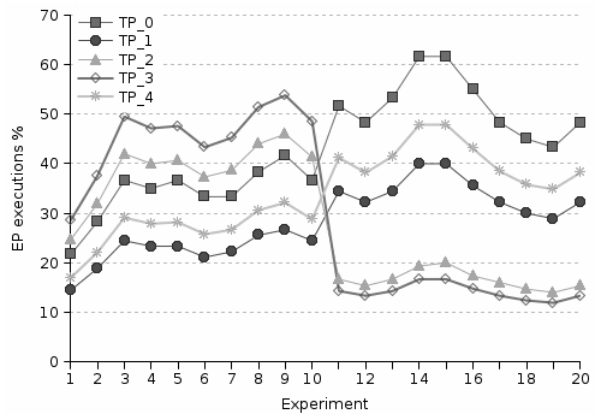

Figure 5. Evaluation while dynamically changing execution *modes*

Description of execution *modes* consists in the simpler way to resolve the situation. It is considered that the system has a limited number of modes, each one composed of fixed *values* for each task. This simulation counts with only two different modes. The first is the initial one, as showed in Figure 5 until half of simulation. At this point the second mode is used, and its activation is clearly visible in the results. In the second mode `TP_2` and `TP_3` are the TPs with greater *value*, and present the lower EPs executions, reaching the proposed goal.

5 Conclusions

This work focused on finding a solution for two issues commonly found in dynamic real-time mobile applications: changing tasks *value* or importance according to the application status and properly handling timing faults in overload situations. It also searched a mechanism that allows the prediction of timing faults and consequently support a dynamic tuning of such faults. As its main contribution, the paper presented a new scheduling algorithm called DMB (Dynamic Misses Based) to dynamically change tasks *value* for adjusting their timing faults rate according to its importance. The performance of DMB is compared with other related algorithms in the performed evaluation. Two different analyzes were performed, one using constant *values* for tasks and another with dynamic (changing) *values* for tasks.

Simulation results with constant *values* show that the scheduling algorithms that use the *value* parameter present different behavior regarding the order in which tasks are discarded (or stop responding to the system) or even to determine the relation between tasks timing faults. In conjunction with TAFT, the DMB algorithm reached the most promising results, since it is able to control tasks degradation (timing faults rate) of a large number of tasks in a graceful and determined way along execution. Finally, although solutions like HDF could also be combined with TAFT to improve performance, it would remain the drawback of losing the lowest *valuable* tasks.

For future work we should explore a more precise way to define values for the application tasks. Moreover, we intend to implement TAFT with DMB in the adopted testbed for mobile applications - the RoboCup - to confirm the results presented in this paper.

Acknowledgments

This project has been financially supported by CAPES, CNPq, and FAPESC (Brazilian Research agencies), and also by the German Academic Exchange Service (DAAD).

References

[1] L. B. Becker, E. Nett, S. Schemmer, and M. Gergeleit. Robust scheduling in team-robotics. *Journal of Systems and Software*, 77(1):3–16, 2005.

[2] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[3] G. C. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *IEEE Real-Time Systems Symposium*, pages 90–99, 1995.

[4] R. K. Clark. *Scheduling dependent real-time activities*. PhD thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, August 1990.

[5] R. K. Clark, E. D. Jensen, and N. F. Rouquette. Software organization to facilitate dynamic processor scheduling. In *IEEE Workshop on Parallel and Distributed Real-Time Systems*, 2004.

[6] M. Gergeleit and H. Streich. Taskpair-scheduling with optimistic case execution times - an example for an adaptive real-time system. In *Second Workshop on Object-Oriented Real-Time Dependable Systems, 1996. Proceedings of WORDS '96*, pages 69–76, 1996.

[7] E. Jensen, C. Locke, and H. Tokuda. A time driven scheduling model for real-time operating systems. In *Proceedings IEEE Real-Time Sytems Symposium*, pages 112–122, 1985.

[8] E. Nett, M. Gergeleit, and M. Mock. Enhancing o-o middleware to become time-aware. *Real-Time Syst.*, 20(2):211–228, 2001.

[9] E. Nett and H. Streich. The gmd-snake - real-time scheduling of a flexible robot application at run-time. In *International Workshop on Parallel Computations and Scheduling*, 1997.

[10] S. Schemmer and E. Nett. Achieving reliable and timely task execution in mobile embedded applications. *Int. Journal of Computer System Science and Engineering*, 2005.

[11] N. Weiderman. Hartstone: Synthetic benchmark requirements for hard real-time applications. In *Technical Report CMU/SEI-89-TR-023*, 1989.